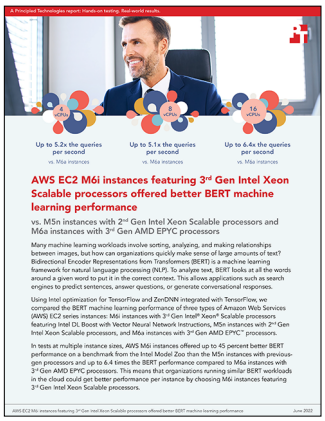

The science behind the report:

AWS EC2 M6i instances featuring 3rd Gen Intel Xeon Scalable processors offered better BERT machine learning performance

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report AWS EC2 M6i instances featuring 3rd Gen Intel Xeon Scalable processors offered better BERT machine learning performance.

We concluded our hands-on testing on April 4, 2022. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on April 1, 2022 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Throughput (examples/sec) for the instances using batch size 1.

	xlarge	2xlarge	4xlarge	8xlarge	16xlarge
m6i (INT8)	2.54	4.7	9.74	13.19	16.86
m6a (FP32)	0.48	0.92	1.52	2.2	2.59
m5n (INT8)	2.14	4.2	6.68	10.05	10.44
m6i (FP32)	0.81	1.51	2.76	4.45	5.31
m5n (FP32)	0.67	1.28	2.33	3.83	4.49

Table 2: Throughput (examples/sec) for the instances using batch size 32.

	xlarge	2xlarge	4xlarge	8xlarge	16xlarge
m6i (INT8)	2.59	4.84	9.82	15.29	22.7
m6a (FP32)	0.61	1.09	1.54	2.96	3.95
m5n (INT8)	2.29	4.36	8.07	12.41	18.65
m6i (FP32)	0.96	1.84	3.45	6.05	8.75
m5n (FP32)	0.79	1.58	2.92	4.78	6.77

System configuration information

Table 3: Detailed information about the M5n instances we tested.

System configuration information	m5n.xlarge	m5n.2xlarge	m5n.4xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	04/04/2022	04/04/2022	04/04/2022
CSP / region	us-east-1d	us-east-1d	us-east-1d
Workload	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)
Workload-specific parameters	Sequence Length: 384	Sequence Length: 384	Sequence Length: 384
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m5n.xlarge	m5n.2xlarge	m5n.4xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Ubuntu 20.04 LTS 5.13.0-1019-aws	Ubuntu 20.04 LTS 5.13.0-1019-aws	Ubuntu 20.04 LTS 5.13.0-1019-aws
Date of last OS updates/ patches applied	04/01/2022	04/01/2022	04/01/2022
Processor			
Number of processors	1	1	1
Vendor and model	Intel® Xeon® 8259CL	Intel Xeon 8259CL	Intel Xeon 8259CL
Core count (per processor)	24	24	24
Core frequency (GHz)	2.50	2.50	2.50
Stepping	7	7	7
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	4	8	16
Memory module(s)			
Total memory in system (GB)	16	32	64
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	16	32	64
General HW			
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance	Direct Att / Instance
Local storage			
Number of drives	1	1	1
Drive size (GB)	50	50	50
Drive information (speed, interface, type)	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x (Up to 25 Gbps)	1x (Up to 25 Gbps)	1x (Up to 25 Gbps)

Table 4: Detailed information about the M6a instances we tested.

System configuration information	m6a.xlarge	m6a.2xlarge	m6a.4xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	04/04/2022	04/04/2022	04/04/2022
CSP / region	us-east-1d	us-east-1d	us-east-1d
Workload	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)
Workload-specific parameters	Sequence Length: 384	Sequence Length: 384	Sequence Length: 384
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m6a.xlarge	m6a.2xlarge	m6a.4xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Ubuntu 20.04 LTS 5.13.0-1019-aws	Ubuntu 20.04 LTS 5.13.0-1019-aws	Ubuntu 20.04 LTS 5.13.0-1019-aws
Date of last OS updates/ patches applied	04/01/2022	04/01/2022	04/01/2022
Processor			
Number of processors	1	1	1
Vendor and model	AMD EPYC 7R13 processor	AMD EPYC 7R13 processor	AMD EPYC 7R13 processor
Core count (per processor)	32	32	32
Core frequency (GHz)	2.62	2.62	2.62
Stepping	1	1	1
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	4	8	16
Memory module(s)			
Total memory in system (GB)	16	32	64
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	16	32	64
General HW			
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance	Direct Att / Instance
Local storage			
Number of drives	1	1	1
Drive size (GB)	50	50	50
Drive information (speed, interface, type)	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x (Up to 12.5 Gbps)	1x (Up to 12.5 Gbps)	1x (Up to 12.5 Gbps)

Table 5: Detailed information about the M6i instances we tested.

System configuration information	m6i.xlarge	m6i.2xlarge	m6i.4xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	04/04/2022	04/04/2022	04/04/2022
CSP / region	us-east-1d	us-east-1d	us-east-1d
Workload	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)	BERT-Large, Uncased (Whole Word Masking)
Workload-specific parameters	Sequence Length: 384	Sequence Length: 384	Sequence Length: 384
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m6i.xlarge	m6i.2xlarge	m6i.4xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Ubuntu 20.04 LTS 5.13.0-1019-aws	Ubuntu 20.04 LTS 5.13.0-1019-aws	Ubuntu 20.04 LTS 5.13.0-1019-aws
Date of last OS updates/ patches applied	04/01/2022	04/01/2022	04/01/2022
Processor			
Number of processors	1	1	1
Vendor and model	Intel Xeon Platinum 8375C	Intel Xeon Platinum 8375C	Intel Xeon Platinum 8375C
Core count (per processor)	32	32	32
Core frequency (GHz)	2.90	2.90	2.90
Stepping	6	6	6
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	4	8	16
Memory module(s)			
Total memory in system (GB)	16	32	64
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	16	32	64
General HW			
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance	Direct Att / Instance
Local storage			
Number of drives	1	1	1
Drive size (GB)	50	50	50
Drive information (speed, interface, type)	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS	gp3, EBS, 3000 IOPS
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x (Up to 12.5 Gbps)	1x (Up to 12.5 Gbps)	1x (Up to 12.5 Gbps)

How we tested

Testing overview

We tested three types of AWS instances: M6i instances with 3rd Gen Intel Xeon Scalable processors, M5i instances with 2nd Gen Intel Xeon Scalable processors, and M6a instances with 3rd Gen AMD EPYC processors. We ran a BERT workload on the AWS instances to show the performance difference between these instance types in terms of examples/second.

Creating the AWS instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, and from the drop-down menu, click Launch instance to open the Launch Instance wizard.
4. In the search window, enter Ubuntu, and press Enter.
5. On the Quick Start tab, select the button next to Ubuntu Server 20.04 LTS (HVM), SSD-Volume Type.
6. On the Choose Instance Type tab, select {m6i,m6a,m5n}.{xlarge, 2xlarge, 4xlarge}. Click Next: Configure Instance Details.
7. On the Configure Instance tab, set the following:
   - Number of instances: 1
   - Purchasing option: Leave unchecked
   - Network: Default VPC
   - Subnet: Choose the region you're working in (we chose us-east-1d)
   - Auto-assign Public IP: Enable
   - Placement Group: Leave unchecked
   - Capacity Reservation: Open
   - Domain join directory: No Directory
   - IAM role: None
   - Shutdown behavior: Stop
8. Click Next: Add Storage.
9. On the Add Storage tab, set the following:
   - Size: 50GB
   - Volume Type: gp3
   - Delete on Termination: Checked
   - Encryption: Not Encrypted
10. Click Next: Add Tags.
11. On the Add Tags tab, add any appropriate tags, and click Next: Configure Security Group.
12. On the Configure Security Group tab, set the following:
    a. Create a new security group.
    b. Leave the rules default.
    c. Click Review, and Launch.
13. On the Review Tab, click Launch.
14. Choose the appropriate option for the key pair, and click Launch Instances.

Configuring Ubuntu 20.04 LTS for BERT benchmark

1. Log into the AWS instance via SSH.
2. Install updates, and reboot the instance:

```
sudo apt-get update
sudo apt-get upgrade -y
sudo reboot
```

3. Install new tools:

```
sudo apt-get autoremove -y
sudo apt-get install -y build-essential numactl hwloc zip unzip sysstat
```

4. Install Anaconda. Accept the license agreement when prompted, choose the default installation location, and answer Yes when prompted to initialize Anaconda using conda init.

```
wget https://repo.anaconda.com/archive/Anaconda3-2021.11-Linux-x86_64.sh
bash Anaconda3-2021.11-Linux-x86_64.sh
```

5. Install and configure libtcmalloc:

```
sudo apt-get install -y libgoogle-perftools4
sudo ln -s /lib/x86_64-linux-gnu/libtcmalloc.so.4 /usr/lib/libtcmalloc.so
```

6. Install and configure docker-ce:

```
sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common
gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
sudo apt-get install -y docker-ce
sudo usermod -aG docker ${USER}
```

7. Log out of the AWS instance, then log back in via ssh to enable conda and docker environment variables.

Installing ZenDNN v3.2 TensorFlow v2.7

1. Download the TF_v2.7_ZenDNN_v3.2_Python_v3.8.zip bundle from https://developer.amd.com/zendnn/.
2. Extract the downloaded archive:

```
unzip TF_v2.7_ZenDNN_v3.2_Python_v3.8.zip
```

3. Install ZenDNN v3.2 TensorFlow v2.7:

```
cd TF_v2.7_ZenDNN_v3.2_Python_v*/
source scripts/TF_ZenDNN_setup_release.sh
```

4. This will automatically create a new conda environment named tf-2.7-zendnn-v3.2-rel-env.

Installing Intel TensorFlow v2.7

1. Create a new conda environment with Python v3.8 to use with Intel TensorFlow v2.7, and activate it:

```
conda create -n tf-2.7-intel python=3.8 -y
conda activate tf-2.7-intel
```

2. Install intel-tensorflow-2.7 in the conda environment by running the following command:

```
pip install intel-tensorflow==2.7
```

Downloading the BERT benchmark and models

1. Make a directory to hold the benchmark and all model files:

```
mkdir /bert
cd /bert
```

2. Download the BERT benchmark from Intel Model Zoo git repo:

```
git clone https://github.com/IntelAI/models.git
git -C models checkout 01839eaa
```

3. Patch the benchmark to include INT8 support (using patch file included in Appendix):

```
mkdir models/benchmarks/language_modeling/tensorflow/bert_large/inference/int8
cd models/benchmarks/language_modeling/tensorflow/bert_large/inference/int8
wget https://raw.githubusercontent.com/IntelAI/models/icx-launch-public/benchmarks/language_
modeling/tensorflow/bert_large/inference/int8/config.json
wget https://raw.githubusercontent.com/IntelAI/models/icx-launch-public/benchmarks/language_
modeling/tensorflow/bert_large/inference/int8/model_init.py
patch model_init.py int8_model_init.py.patch
cd /bert
```

4. Download and unzip the BERT large uncased (whole word masking) model from the Google BERT repo:

```
wget https://storage.googleapis.com/bert_models/2019_05_30/wwm_uncased_L-24_H-1024_A-16.zip
unzip wwm_uncased_L-24_H-1024_A-16.zip
```

5. Download the dev-v1.1.json file from the Google BERT repo into the wwm_uncased_L-24_H-1024_A-16 directory that you just unzipped:

```
wget https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-v1.1.json -P wwm_
uncased_L-24_H-1024_A-16
```

6. Download and unzip the pretrained model checkpoint files:

```
wget https://storage.googleapis.com/intel-optimized-tensorflow/models/v1_8/bert_large_
checkpoints.zip
unzip bert_large_checkpoints.zip
```

7. Download the FP32 frozen graph:

```
wget https://storage.googleapis.com/intel-optimized-tensorflow/models/v2_7_0/fp32_bert_squad.pb
```

8. Download the INT8 frozen graph:

```
wget https://storage.googleapis.com/intel-optimized-tensorflow/models/r2.5-icx-b631821f/
asymmetric_per_channel_bert_int8.pb
```

Running the BERT benchmark

If using FP32, activate the conda environment for either ZenDNN or Intel TensorFlow:

- Intel TensorFlow:

```
conda activate tf-2.7-intel
```

- ZenDNN TensorFlow:

```
conda activate tf-2.7-zendnn-v3.2-rel-env
```

1. Run the conda FP32 script:

```
./run_bert_conda.sh
```

2. If using INT8, then simply running the INT8 script and this will automatically download and create the docker container needed:

```
./run_bert_int8.sh
```

3. Save the throughput results.

run_bert_conda.sh:

```
#!/bin/bash
cd /bert/models/benchmarks
echo ${CONDA_DEFAULT_ENV}

export OUTPUT_DIR=/bert/output_$(curl -q http://169.254.169.254/latest/meta-data/instance-type 2>/
dev/null)_${CONDA_DEFAULT_ENV}
export CHECKPOINT_DIR=/bert/bert_large_checkpoints
export DATASET_DIR=/bert/wwm_uncased_L-24_H-1024_A-16
export PRETRAINED_MODEL=/bert/fp32_bert_squad.pb

mkdir -p ${OUTPUT_DIR}

time python launch_benchmark.py \
    --model-name=bert_large \
    --precision=fp32 \
    --mode=inference \
    --framework=tensorflow \
    --batch-size=1 \
    --data-location ${DATASET_DIR} \
    --in-graph ${PRETRAINED_MODEL} \
    --checkpoint ${CHECKPOINT_DIR} \
    --output-dir ${OUTPUT_DIR} \
    --benchmark-only \
    --warmup-steps=30 \
    --steps=90 \
    --infer_option=SQuAD

time python launch_benchmark.py \
    --model-name=bert_large \
    --precision=fp32 \
    --mode=inference \
    --framework=tensorflow \
    --batch-size=32 \
    --data-location ${DATASET_DIR} \
    --in-graph ${PRETRAINED_MODEL} \
    --checkpoint ${CHECKPOINT_DIR} \
```

```
    --output-dir ${OUTPUT_DIR} \
    --benchmark-only \
    --warmup-steps=3 \
    --steps=9 \
    --infer_option=SQuAD

grep -B1 -A2 "Throughput: " ${OUTPUT_DIR}/*.log
```

run_bert_int8.sh:

```
#!/bin/bash
cd /bert/models/benchmarks
DOCKER_IMAGE=tf-r2.5-icx-b631821f
echo ${DOCKER_IMAGE}

export OUTPUT_DIR=/bert/output_$(curl -q http://169.254.169.254/latest/meta-data/instance-type 2>/
dev/null)_${DOCKER_IMAGE}
export CHECKPOINT_DIR=/bert/bert_large_checkpoints
export DATASET_DIR=/bert/wwm_uncased_L-24_H-1024_A-16
export PRETRAINED_MODEL=/bert/asymmetric_per_channel_bert_int8.pb

mkdir -p ${OUTPUT_DIR}

cd /bert/models/benchmarks
time python launch_benchmark.py \
    --model-name=bert_large \
    --precision=int8 \
    --mode=inference \
    --framework=tensorflow \
    --batch-size=1 \
    --data-location ${DATASET_DIR} \
    --in-graph ${PRETRAINED_MODEL} \
    --checkpoint ${CHECKPOINT_DIR} \
    --output-dir ${OUTPUT_DIR} \
    --benchmark-only \
    --warmup-steps=30 \
    --steps=90 \
    --docker-image intel/intel-optimized-tensorflow:${DOCKER_IMAGE} \
    --infer_option=SQuAD

time python launch_benchmark.py \
    --model-name=bert_large \
    --precision=int8 \
    --mode=inference \
    --framework=tensorflow \
    --batch-size=32 \
    --data-location ${DATASET_DIR} \
    --in-graph ${PRETRAINED_MODEL} \
    --checkpoint ${CHECKPOINT_DIR} \
    --output-dir ${OUTPUT_DIR} \
    --benchmark-only \
    --warmup-steps=3 \
    --steps=9 \
    --docker-image intel/intel-optimized-tensorflow:${DOCKER_IMAGE} \
    --infer_option=SQuAD

grep -B1 -A2 "Throughput: " ${OUTPUT_DIR}/*.log
```

int8_model_init.py.patch:

```
--- model_init.py    2022-04-09 06:26:14.335506293 -0400
+++ int8_model_init.py      2022-03-07 16:26:12.305330529 -0500
@@ -25,6 +25,7 @@
 from common.base_model_init import set_env_var

 import os
+import sys
 from argparse import ArgumentParser


@@ -38,6 +39,24 @@
         if not platform_util:
             raise ValueError("Did not find any platform info.")

+        # multi-instance runs on bert require each instance to have separate output folders
+        # and the output folders should be empty to prevent conflicts with previous runs
+        if self.args.numa_cores_per_instance and os.path.exists(
+                self.args.output_dir):
+            if os.listdir(self.args.output_dir):
+                sys.exit(
+                    "ERROR: The output directory ({}) is not empty. BERT multi-instance "
+                    "runs require empty output directories in order to prevent conflicts "
+                    "with files generated by previous runs. Please provide an empty folder "
+                    "using the --output-dir argument.".format(self.args.output_dir))
+
+        # BERT FP32 inference requires a frozen graph
+        if not self.args.input_graph:
+            sys.exit("ERROR: BERT large FP32 inference requires a frozen graph to be passed "
+                     "to the launch_benchmark.py script using the --in-graph arg. Please download "
+                     "the frozen graph using the instructions in the README.md and
update your command "
+                     "to add the input graph.")
+
         # use default batch size of 32 if it's -1
         if self.args.batch_size == -1:
             self.args.batch_size = 32
@@ -63,6 +82,12 @@
             "--experimental-gelu", dest="experimental_gelu", default="False")
         arg_parser.add_argument(
             "--optimized-softmax", dest="optimized_softmax", default="True")
+        arg_parser.add_argument("--warmup-steps", dest='warmup_steps',
+                                type=int, default=10,
+                                help="number of warmup steps")
+        arg_parser.add_argument("--steps", dest='steps',
+                                type=int, default=30,
+                                help="number of benchmark steps")

         self.args = arg_parser.parse_args(self.custom_args, namespace=self.args)

@@ -133,6 +158,12 @@
         if self.args.num_intra_threads:
             model_args += " --intra_op_parallelism_threads=" + str(self.args.num_intra_threads)

+        if self.args.warmup_steps:
+            model_args += " --warmup_steps=" + str(self.args.warmup_steps)
+
+        if self.args.steps:
+            model_args += " --steps=" + str(self.args.steps)
+
         self.benchmark_command = self.get_command_prefix(args.socket_id) + \
             self.python_exe + " " + model_script + model_args
```

Read the report at https://facts.pt/HnKi6oO ▶

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®